简易数据存储系统说明文档

姓名：周鑫

学号：515030910073

班级：515030910073

目录

一、概述

该程序属于“程序设计与数据结构”暑期课程设计大作业，要求实现具有增删改查功能的可存储<key, value>对的数据库。在该程序中 key、value 均为少于 15 字符的字符。设计的类 dataBase 中所含有的公有成员函数如下：

dataBase(string &path); 打开一个数据库并初始化，路径地址为 path;
dataBase(); 打开一个数据库但暂不初始化;
void open(string &path); 对一个未初始化的数据库进行初始化;
void store(string &key, string &value); 在数据库中插入一个 key, value 对;
void del(string &key); 从数据库中删除一个 key, value 对;
string find(string &key); 从数据库中查找一个 key 对应的 value;
void close(); 关闭数据库。

二、数据结构

该程序所采用的数据结构为散列表，将散列表中每个桶存入索引文件中，并采用多槽位法和线性试探法解决冲突问题。

三、具体实现

1. 文件分配

数据库文件分为两个，索引文件和数据文件。索引文件中存放自定义的结构体 bucket，其中包含构成散列表所需的成员；数据文件中存放大小为 15 的字符数组。

2. 结构体

程序中自定义了名为 bucket 的结构体，其中包含构成散列表所需的成员 :keys, values, bucketN。其中，keys 是大小为 10 的二维字符数组，对应该桶中的 10 个槽位；values 是大小为 10 的整数数组，存放 value 在数据文件中的偏移量 ;bucketN 是该桶的编号(0~700000)。

3. 类

程序中自定义了名为 dataBase 的类，其公有成员函数包含增删改查、打开关闭数据库的功能，私有变量为文件流 idx 和 dat，分别对应索引文件和数据文件。其中各个公有成员函数依赖于私有函数实现，私有函数功能如下：

int hashFunc(string &key); 该函数对输入的字符串 key 进行哈希，获得桶的编号。

void rehash(int &bucketN); 该函数将桶编号再次哈希（由于自己写的重散列函数可能不够成熟，后改为直接将 bucketN 加一，采用线性试探法解决冲突）。

void idx_find_ins(string &key, int &ps, int &bucketN, bucket &b); 该函数根据桶编号在索引文件中读出对应的桶保存于 b 中，并且找到可能可用的 value 在数据文件中的插入位置并存入 ps 中（如桶内原有相同的 key 则将 value 存入原有 value 的位置，如桶内有被删除过的 key 则将 value 存入被删除的 value 的位置）。

int ins_check(bucket &b, string &key); 该函数检查桶 b 中是否有对应的 key 或空闲槽位，并返回槽位编号（若无则返回-1）。

int dat_ins(string &value, int &ps); 该函数根据 ps（可能可用的插入位置）将 value 插入至数据文件中，并返回 value 的实际插入位置。

void idx_ins(string &key, int &bucketN, int &p, bucket &b, int &last); 该函数根据 value 的插入位置 p 将桶 b 的 values 更改，并将更改过的桶 b 写入索引文件对应桶位置。

int idx_find(string &key, int &bucketN); 该函数根据桶编号找到 key 对应的 value 在数据文件中的位置。

string dat_find(int &p); 该函数找到数据文件中位于 p 的 value。

bool idx_delete(string &key, int &bucketN, int &p); 该函数根据桶编号在索引文件中找到 key 对应的 value 的位置保存于 p 中，将桶中 key 更改为空字符串并返回 true（若没有找到，返回 false）。

4. 空闲空间管理

程序中每次删除数据，在数据文件中由于每个 value 空间固定为 15(以字符数组形式存储)，故并不对 value 进行修改，而只是在索引文件中将对应的桶中对应 key 修改为空字符串，在下次插入至该桶中时调用 idx_find_ins 函数可以将该 key 对应的 value 位置找到再次利用，以免数据文件大小增长过快。

四、测试

1. 正确性测试

测试方法：分别向数据库和 C++的 map 中插入 100 条数据，再从数据中和 map 中查找这 100 条数据，若同一 key 查找到的 key 不同则输出错误；将 100 条数据从数据库中删除过后再次查找这 100 条数据，若不是空字符串则输出错误。(TEST 0)

测试结果：程序输出符合预期。

2. 性能测试

测试一：插入 nrec 条数据（TEST 1）

结果：nrec = 10000 时用时 0.175s；nrec = 100000 时用时 1.718s；nrec = 1000000 时用时 16.903s

测试二：删除 nrec 条已存在的数据（TEST 2）

结果：nrec = 1000000 时用时 11.177s；nrec = 100000 时用时 1.058s；nrec = 10000 时用时 0.11s

测试三：查找 nrec 条已存在的数据（TEST 3）

结果：nrec = 10000 时用时 0.087s；nrec = 100000 时用时 0.833s；nrec = 1000000 时用时 8.385s

测试四：打开一个不存在的数据库（测试初始化时间）

结果：0.284s

3. 分析

根据正确性测试可知该数据库运行正常。根据性能测试可知，该数据库对百万级数据量的插入速度尚可，对已存在的数据的查找与删除速度也尚可。

五、反思与总结

反思：尽管程序功能实现正确，效率尚可，但由于设计的散列表固定的桶数过多，索引文件大小达到了 130M 之多，在只插入较少数据时索引文件显得太大。据说使用 B+树可解决这个问题，但为了实现方便我还是选择了散列表作为数据结构。另外，使用多槽位法和线性试探法解决冲突也只使得数据库最大存储量达到 7000010 对数据，如果使用链表操作可能可以使数据库可存储的数据量更多。

总结：由于个人水平有限，即使使用散列表作为数据结构完成这次作业也花费了近一周的时间。一开始花一两天实现的是一个 key 为整数，value 为字符串的版本，考虑的东西比较少，解决冲突只使用多槽位法，甚至没有考虑到空闲空间管理，还是室友提到我才想起来。后来感觉这样略 low，就在原来的基础上实现了 key 为字符串的版本。一开始设置二维字符数组时为了让索引文件看上去不至于太大，只设置了 5 个槽位，但后来发现冲突很容易充满每个槽位，就加入了一个重散列的函数；仔细想想自己写的重散列效果可能不太好，干脆又换成了用线性试探法解决冲突。但在使用线性试探法之后，发现在查找/删除不存在的 key 时会花费大量时间，甚至有一个 key 需要两三秒的情况，才想起来自己在这部分写的判断有问题，每次查找一个不存在的 key 要从它 hash 到的那个桶一直检查到散列表中最后一个桶，导致耗时太多。几经修改，才得到现在这个比较满意的版本（尽管还是有缺陷）。因为花在 debug 上的时间比从零写起的时间多得多，这次作业的完成也许可以称为“基于 debug 的程序设计”了（笑）。总之，能够顺利完成这次作业，需要感谢软院同学每天对此的讨论给我的鞭策，需要感谢室友在二进制文件读写方面给我的提示，需要感谢助教们的讲解，也需要感谢老师没有布置更高要求的作业（笑）。